2024 OOP&DS Homework 2

112550013 周廷威

1. Implementation Details

```cpp
#include <bits/stdc++.h>

#define int long long
#define endl '\n'
#define For(z, x, y) for(int z = x; z <= y; z++)
using namespace std;

struct Node{
    int in, w, s;
    vector<int> out;
    Node(): in(0), w(0), s(0){}
} g[1005];

int32_t main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);

    int n, m;
    queue<int> q;
    cin >> n >> m;

    For(i, 1, m){
        int x, y;
        cin >> x >> y;
        g[x].out.push_back(y);
        g[y].in++;
    }
    For(i, 1, n){
        cin >> g[i].w;
        g[i].s = g[i].w;
        if(g[i].in == 0) q.push(i);
    }

    while(!q.empty()){
        int now = q.front();
        q.pop();
        for(auto i: g[now].out){
            g[i].s = max(g[i].s, g[now].s + g[i].w);
            g[i].in--;
            if(g[i].in == 0) q.push(i);
```

```
41          }
42      }
43
44      int ans = 0;
45      For(i, 1, n) ans = max(ans, g[i].s);
46      cout << ans << endl;
47
48      return 0;
49  }
```

(a) Steps of the implementation
The C++ program is designed to solve a problem where you have `n` nodes and `m` directed edges. The goal is to calculate the maximum "weight" that can be obtained by traversing the nodes from start to finish, considering that some nodes can only be visited after certain others (a topological sort scenario).

(b) The way you deal with the data in the problem

- Reading `n` (number of nodes) and `m` (number of directed edges).
- Reading each directed edge and adjusting the adjacency list and the in-degree of each node accordingly.
- Reading weights for each node and initializing the start weights for nodes with zero in-degrees (indicating no prerequisites).

(c) Explain your code

- The `Node` struct holds data for each node: `in` (number of prerequisites), `w` (weight of the node), `s` (maximum time up to this node), and `out` (list of nodes this one points to).
- Input the number of nodes (`n`) and directed edges (`m`). Initialize a queue to hold nodes that are ready to be processed (no remaining prerequisites).
- For each edge from `x` to `y`, increment the in-degree of `y` to indicate `y` depends on `x`, and append `y` to the outgoing list of `x`.
- For each node, input its weight and if it has no incoming edges (in-degree zero), add it to the queue with its time set to its weight.
- Process the queue:
  - Pop a node from the queue, for each of its outgoing nodes, update the time if traversing from the current node gives a higher time.
  - Decrease the in-degree of the outgoing nodes. If any reaches zero, add it to the queue.
- After processing all nodes, compute the maximum time found across all nodes.

2. Time Complexity

- The overall time complexity is $O(n + m)$, suitable for handling large data efficiently under the constraints given.

3. Discussion

(a) Your discover

- Discovered that using topological sorting combined with dynamic programming can efficiently solve dependency and scheduling problems.

(b) Which is better algorithm in which condition

- This algorithm excels in scenarios where the graph has a clear topological order, such as scheduling and project planning tasks.

(c) Challenges you encountered

- Managing and debugging the dependencies and ensuring all nodes are processed in the correct order was challenging.